Basic Workflow of the Notebook

1. Feature Engineering

```
        1.1 Static Data Analysis & Imputation
        1.2 Temporal Data Analysis
        1.3 Feature Extraction from Temporal Data & Imputation
        1.4 Generation of .csv ( contains above three steps )
```

2. Model Deployment

```
        2.1 Classification Model
        2.2 Sensitivity and Specifity Analysis
        2.3 Regression Model and mean RMSE
```

In [111]:
```
import random
import numpy as np
import pandas as pd
import os

#-----Plotting
import matplotlib.pyplot as plt
import seaborn as sns

#-----Pre-processing
from sklearn.preprocessing import Imputer, StandardScaler,OneHotEncoder
from sklearn.decomposition import PCA
from sklearn.datasets import make_classification
from sklearn.feature_selection import SelectKBest,chi2,VarianceThreshold
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.decomposition import TruncatedSVD

#-----Models
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, cross_val_score, KFold, train_test_
split,RandomizedSearchCV
from sklearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestRegressor
from xgboost.sklearn import XGBRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.linear_model import LogisticRegression

#-----Metrics
from sklearn.metrics import roc_auc_score, precision_score, recall_score, confusion_m
atrix, roc_curve, auc,accuracy_score
from sklearn.metrics import mean_squared_error

#-----Scipy
from scipy import stats

#-----For ignoring warnings
import warnings
warnings.filterwarnings("ignore")
```

In [112]:
```
static_feat = ['RecordID', 'Age', 'Gender', 'Height', 'ICUType', 'Weight']
temporal_feat = ['Albumin',       #----- (g/dL)
                 'ALP',           #----- [Alkaline phosphatase (IU/L)]
                 'ALT',           #----- [Alanine transaminase (IU/L)]
                 'AST',           #----- [Aspartate transaminase (IU/L)]
                 'Bilirubin',     #----- (mg/dL)
                 'BUN',           #----- [Blood urea nitrogen (mg/dL)]
                 'Cholesterol',   #----- (mg/dL)
                 'Creatinine',    #----- [Serum creatinine (mg/dL)]
                 'DiasABP',       #----- [Invasive diastolic arterial blood pressure
 (mmHg)]
                 'FiO2',          #----- [Fractional inspired O2 (0-1)]
                 'GCS',           #----- [Glasgow Coma Score (3-15)]
                 'Glucose',       #----- [Serum glucose (mg/dL)]
                 'HCO3',          #----- [Serum bicarbonate (mmol/L)]
                 'HCT',           #----- [Hematocrit (%)]
                 'HR',            #----- [Heart rate (bpm)]
                 'K',             #----- [Serum potassium (mEq/L)]
                 'Lactate',       #----- (mmol/L)
                 'Mg',            #----- [Serum magnesium (mmol/L)]
                 'MAP',           #----- [Invasive mean arterial blood pressure (mmH
g)]
                 'MechVent',      #----- [Mechanical ventilation respiration (0:fals
e, or 1:true)]
                 'Na',            #----- [Serum sodium (mEq/L)]
                 'NIDiasABP',     #----- [Non-invasive diastolic arterial blood press
ure (mmHg)]
                 'NIMAP',         #----- [Non-invasive mean arterial blood pressure
 (mmHg)]
                 'NISysABP',      #----- [Non-invasive systolic arterial blood pressu
re (mmHg)]
                 'PaCO2',         #----- [partial pressure of arterial CO2 (mmHg)]
                 'PaO2',          #----- [Partial pressure of arterial O2 (mmHg)]
                 'pH',            #----- [Arterial pH (0-14)]
                 'Platelets',     #----- (cells/nL)
                 'RespRate',      #----- [Respiration rate (bpm)]
                 'SaO2',          #----- [O2 saturation in hemoglobin (%)]
                 'SysABP',        #----- [Invasive systolic arterial blood pressure
 (mmHg)]
                 'Temp',          #----- [Temperature (°C)]
                 'TroponinI',     #----- [Troponin-I (μg/L)]
                 'TroponinT',     #----- [Troponin-T (μg/L)]
                 'Urine',         #----- [Urine output (mL)]
                 'WBC',           #----- [White blood cell count (cells/nL)]
                 'Weight']        #----- (kg)
```

File Read

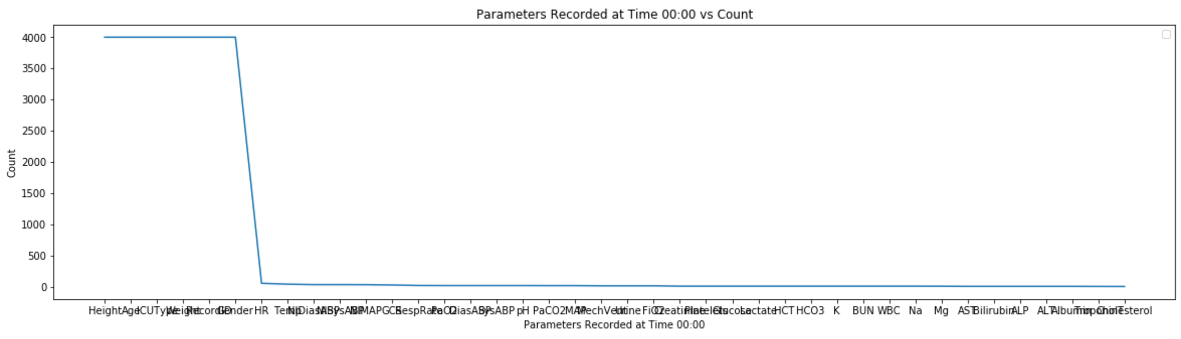
In [132]:
```
#folder = "set-a"
folder = r"predictingChallenge2012\set-a"  #Change folder location
doc = []
count = 0

#----- pre-processing (files - list)
for f in os.listdir(folder):                    #----- Iterate over list of files p
resent in folder
    with open(os.path.join(folder,f),'r') as fp:
        lines = fp.readlines()
    count = count+1
    record_id = lines[1].strip().split(',')[-1]   #----- getting record_id
    doc_dummy = [i.strip().split(',') + [(record_id)] for i in lines]
    doc.extend(doc_dummy[1:])
```

Static Data Analysis & Imputation - Plotting and Extraction of Static Data recorded at T ime 00:00

In [145]:
```
#----- pre-processing (list - dataframe)
df = pd.DataFrame(doc,columns=['Time','Parameter','Value','Record_ID'])
df_static = df.loc[df['Time'] =='00:00']



print("\n1.1) Parameters recorded at time 00:00")
plt.figure(9,figsize = (20,5))
plt.plot(df_static['Parameter'].value_counts())
plt.xlabel("Parameters Recorded at Time 00:00")
plt.ylabel("Count")
plt.title("Parameters Recorded at Time 00:00 vs Count")
plt.legend()
plt.show()



#----- Ensure each patient has only one 'RecordID', 'Age', 'Gender', 'Height', 'ICUTy
pe', 'Weight'
df_static = df_static.loc[df_static['Parameter'].isin(static_feat)]
print("\n1.2) Extracting Static Features alone at Time 00:00")
plt.figure(9,figsize = (20,5))
plt.plot(df_static['Parameter'].value_counts())
plt.xlabel("Parameters Recorded at Time 00:00 After extraction of Static Features")
plt.ylabel("Count")
plt.title("Parameters Recorded at Time 00:00 After extraction of Static Features vs C
ount")
plt.legend()
plt.show()



df = df.loc[~df.index.isin(df_static.index)]      #----- Removing static variables fr
om Temporal Data in df
df_static = df_static.pivot(index='Record_ID', columns='Parameter', values='Value')
print("\n1.3) Number of observations of Static Data\n")
print(df_static.shape)
df_static.head()
```

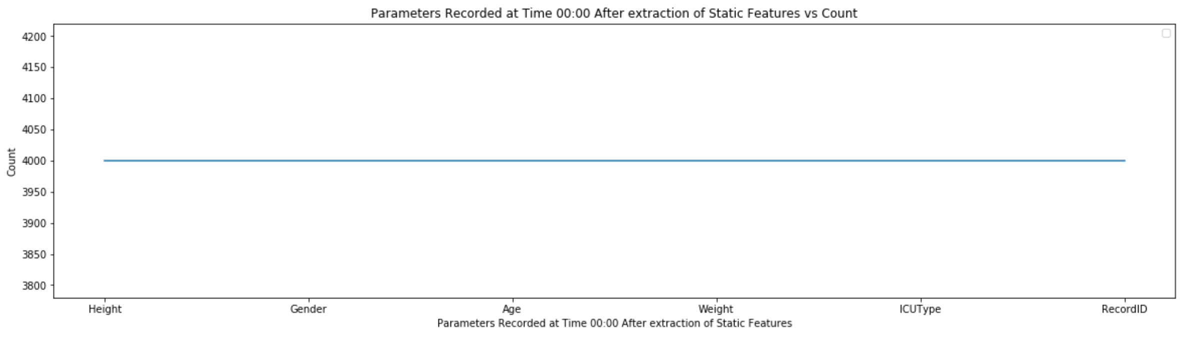
```
No handles with labels found to put in legend.
```

```
1.1) Parameters recorded at time 00:00
```



```
No handles with labels found to put in legend.
```

```
1.2) Extracting Static Features alone at Time 00:00
```



```
1.3) Number of observations of Static Data
```

```
(4000, 6)
```

Out[145]:

Parameter	Age	Gender	Height	ICUType	RecordID	Weight
Record_ID						
132539	54	0	-1	4	132539	-1
132540	76	1	175.3	2	132540	76
132541	44	0	-1	3	132541	56.7
132543	68	1	180.3	3	132543	84.6
132545	88	0	-1	3	132545	-1

In [148]:
```
df_static.info()

df_static
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 4000 entries, 132539 to 142673
Data columns (total 7 columns):
Age          4000 non-null float64
Gender       4000 non-null float64
Height       4000 non-null float64
ICUType      4000 non-null float64
RecordID     4000 non-null float64
Weight       4000 non-null float64
age_group    4000 non-null category
dtypes: category(1), float64(6)
memory usage: 223.0+ KB
```

Out[148]:

Parameter	**Age**	**Gender**	**Height**	**ICUType**	**RecordID**	**Weight**	**age_group**
Record_ID							
132539	54.0	0.0	-1.0	4.0	132539.0	-1.0	50-60
132540	76.0	1.0	175.3	2.0	132540.0	76.0	70-80
132541	44.0	0.0	-1.0	3.0	132541.0	56.7	40-50
132543	68.0	1.0	180.3	3.0	132543.0	84.6	60-70
132545	88.0	0.0	-1.0	3.0	132545.0	-1.0	80-90
132547	64.0	1.0	180.3	1.0	132547.0	114.0	60-70
132548	68.0	0.0	162.6	3.0	132548.0	87.0	60-70
132551	78.0	0.0	162.6	3.0	132551.0	48.4	70-80
132554	64.0	0.0	-1.0	3.0	132554.0	60.7	60-70
132555	74.0	1.0	175.3	2.0	132555.0	66.1	70-80
132556	64.0	0.0	-1.0	3.0	132556.0	65.0	60-70
132567	71.0	0.0	157.5	2.0	132567.0	56.0	70-80
132568	66.0	0.0	157.5	3.0	132568.0	84.5	60-70
132570	84.0	1.0	170.2	1.0	132570.0	102.6	80-90
132573	77.0	1.0	162.6	1.0	132573.0	90.1	70-80
132575	78.0	1.0	167.6	2.0	132575.0	63.0	70-80
132577	65.0	1.0	-1.0	3.0	132577.0	66.3	60-70
132582	84.0	1.0	182.9	3.0	132582.0	82.5	80-90
132584	78.0	0.0	-1.0	3.0	132584.0	72.8	70-80
132585	40.0	0.0	165.1	2.0	132585.0	84.7	40-50
132588	48.0	0.0	154.9	3.0	132588.0	42.3	40-50
132590	58.0	1.0	188.0	2.0	132590.0	98.0	50-60
132591	81.0	1.0	-1.0	3.0	132591.0	63.7	80-90
132592	35.0	0.0	-1.0	3.0	132592.0	71.8	30-40
132595	26.0	0.0	-1.0	3.0	132595.0	-1.0	20-30
132597	66.0	0.0	137.2	3.0	132597.0	82.0	60-70
132598	80.0	0.0	-1.0	4.0	132598.0	60.0	80-90
132599	53.0	0.0	177.8	4.0	132599.0	73.5	50-60
132601	74.0	1.0	177.8	2.0	132601.0	75.9	70-80
132602	80.0	1.0	180.3	3.0	132602.0	70.0	80-90
...	...	...	...	...	...	...	...
142591	69.0	1.0	177.8	2.0	142591.0	104.0	60-70
142595	67.0	1.0	-1.0	3.0	142595.0	68.0	60-70
142601	78.0	0.0	157.5	2.0	142601.0	54.1	70-80
142603	61.0	1.0	182.9	4.0	142603.0	95.0	60-70
142607	60.0	1.0	-1.0	1.0	142607.0	95.3	60-70
142609	38.0	0.0	165.1	2.0	142609.0	80.0	30-40
142612	55.0	0.0	-1.0	3.0	142612.0	109.7	50-60
142618	57.0	1.0	188.0	3.0	142618.0	93.5	50-60
142621	85.0	0.0	-1.0	3.0	142621.0	61.5	80-90
142626	83.0	1.0	180.3	4.0	142626.0	70.0	80-90

Parameter	Age	Gender	Height	ICUType	RecordID	Weight	age_group
Record_ID							
142634	80.0	1.0	167.6	3.0	142634.0	63.9	80-90
142635	67.0	0.0	170.2	4.0	142635.0	118.0	60-70
142637	73.0	0.0	152.4	2.0	142637.0	57.7	70-80
142638	74.0	0.0	-1.0	3.0	142638.0	65.0	70-80
142640	65.0	1.0	-1.0	3.0	142640.0	105.1	60-70
142641	50.0	0.0	-1.0	4.0	142641.0	-1.0	50-60
142646	34.0	1.0	175.3	4.0	142646.0	70.0	30-40
142649	75.0	1.0	-1.0	3.0	142649.0	80.7	70-80
142653	72.0	0.0	165.1	4.0	142653.0	80.0	70-80
142654	66.0	1.0	-1.0	3.0	142654.0	78.2	60-70
142655	43.0	1.0	-1.0	3.0	142655.0	92.9	40-50
142659	88.0	1.0	-1.0	1.0	142659.0	90.7	80-90
142661	89.0	1.0	177.8	4.0	142661.0	64.0	80-90
142662	86.0	1.0	162.6	3.0	142662.0	53.0	80-90
142664	51.0	0.0	-1.0	4.0	142664.0	75.0	50-60
142665	70.0	0.0	-1.0	4.0	142665.0	87.0	70-80
142667	25.0	1.0	-1.0	3.0	142667.0	166.4	20-30
142670	44.0	1.0	-1.0	3.0	142670.0	109.0	40-50
142671	37.0	1.0	-1.0	3.0	142671.0	87.4	30-40
142673	78.0	0.0	157.5	4.0	142673.0	70.7	70-80

4000 rows × 7 columns

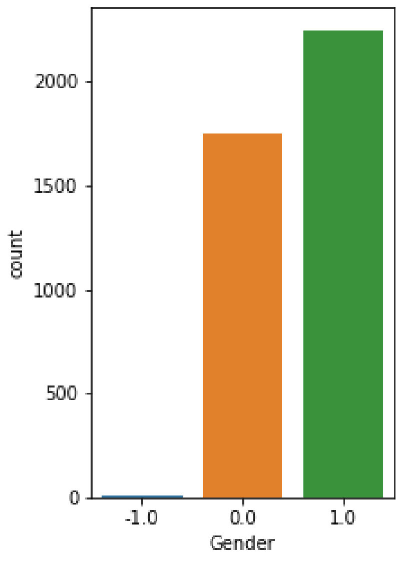
Static Data Exploration for Imputation - Plotting of

1. Gender vs Count
2. Age Vs Count
3. Height & Weight based on its age-group vs Count

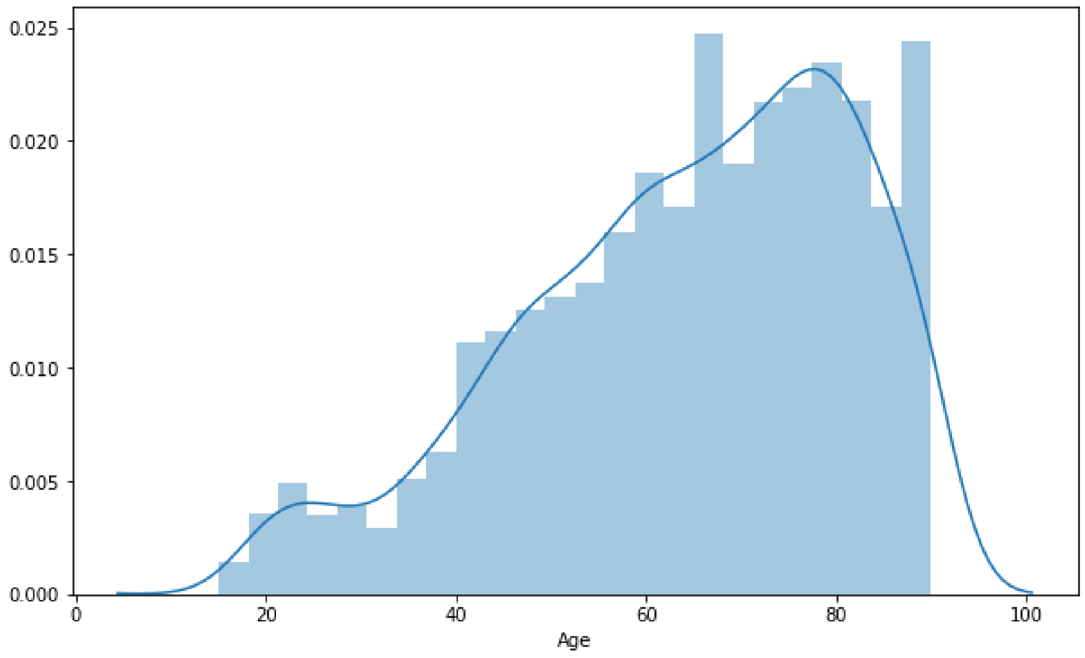
Static data

1. RecordID (a unique integer for each ICU stay)
2. Age (years)
3. Gender (0: female, or 1: male)
4. Height (cm)
5. ICUType (1: Coronary Care Unit, 2: Cardiac Surgery Recovery Unit,3: Medical ICU, or 4: Surgical ICU)
6. Weight (kg)

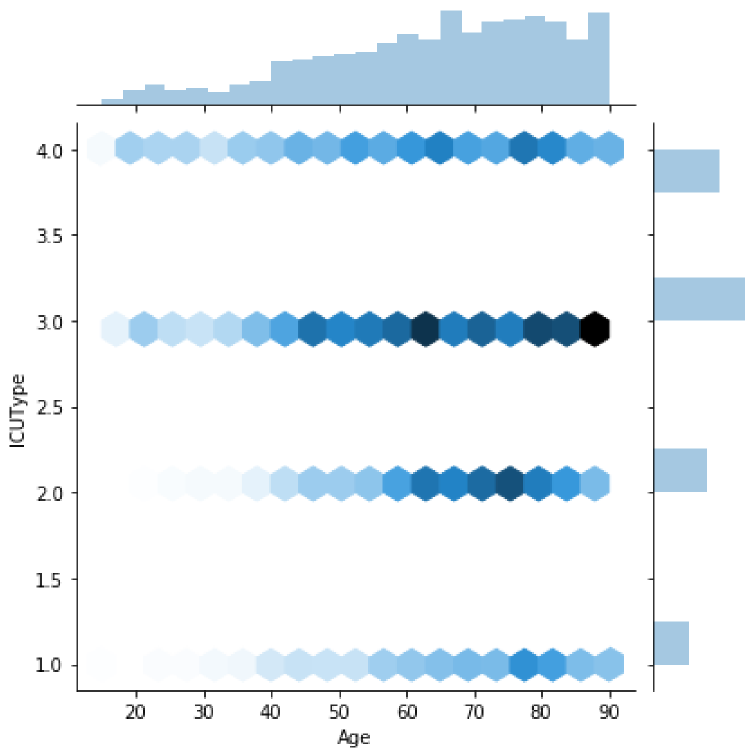
In [147]:
```
#----- converting static into numeric
df_static = df_static.astype(float)

#----- Grouping Height, Weight based on the age-group for imputation
df_static['age_group'] = pd.cut(df_static.Age, [0, 20, 30, 40, 50, 60, 70, 80, 90,flo
at('Inf')], right=False, labels=["<20", "20-30", "30-40","40-50","50-60","60-70","70-
80","80-90",">=90"])


df_static_viz = df_static.reset_index(drop=True)

#----- Analysis of Gender data for imputation
plt.figure(figsize = (3,5))
sns.countplot(x='Gender',data=df_static_viz)
plt.show()

print(df_static['Gender'].value_counts())

#----- Analysis of Age data for imputation
plt.figure(9,figsize = (10,6))
sns.distplot(df_static_viz['Age'])
plt.show()

#ICUType vs Age
plt.figure(figsize = (15,6))
sns.jointplot(x='Age',y='ICUType',data=df_static_viz,kind='hex')
plt.show()

plt.figure(figsize = (15,6))
sns.boxplot(x="age_group", y="Height", data=df_static_viz,palette='rainbow')
plt.show()

plt.figure(figsize = (15,6))
sns.boxplot(x="age_group", y="Weight", data=df_static_viz,palette='rainbow')
plt.show()


df_static_pairplot = df_static_viz.drop(['RecordID','age_group','ICUType'],axis =1)
sns.pairplot(df_static_pairplot,hue='Gender')
```

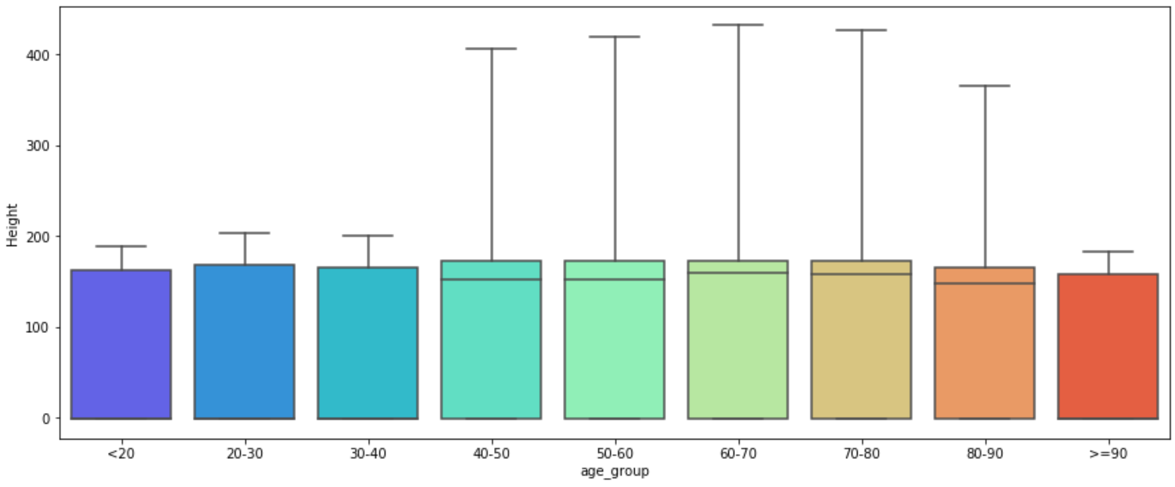


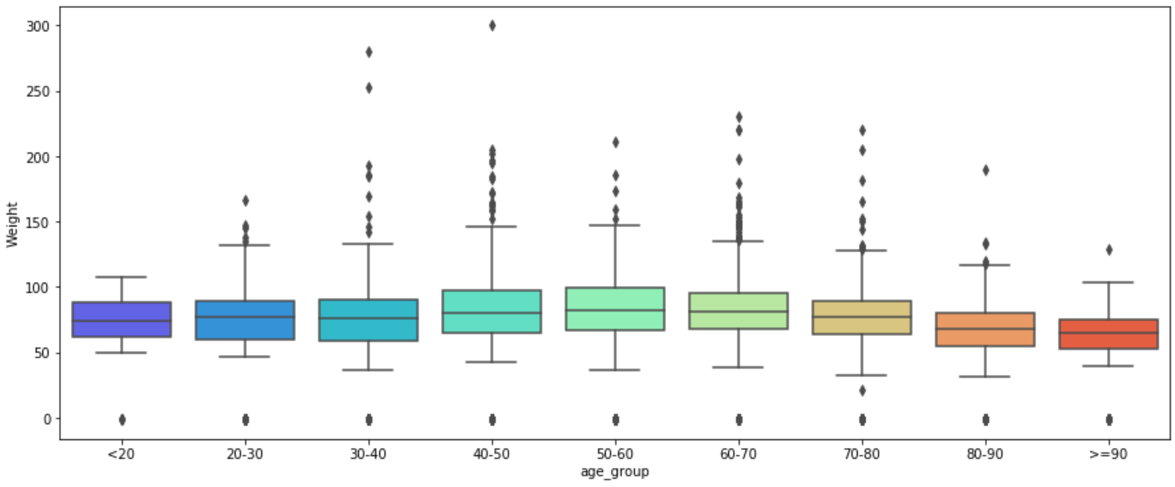
```
 1.0    2246
 0.0    1751
-1.0       3
Name: Gender, dtype: int64
```



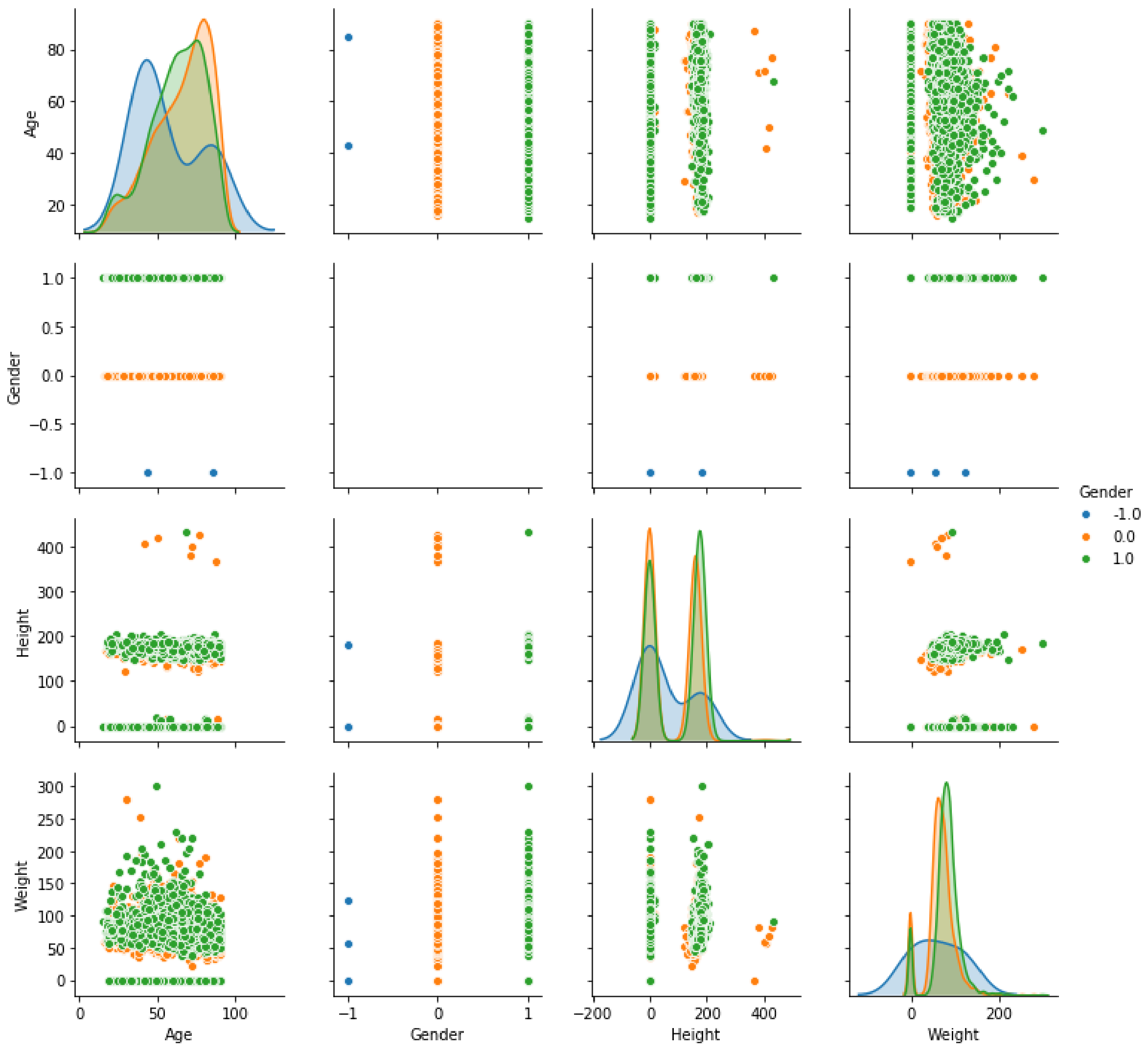
```
<Figure size 1080x432 with 0 Axes>
```


ICUType
4.0
3.5
3.0
2.5
2.0
1.5
1.0
20
30
40
50
60
70
80
90
Age


Height
400
300
200
100
0
<20
20-30
30-40
40-50
50-60
60-70
70-80
80-90
>=90
age_group


Weight
300
250
200
150
100
50
0
<20
20-30
30-40
40-50
50-60
60-70
70-80
80-90
>=90
age_group

```
Out[147]: <seaborn.axisgrid.PairGrid at 0x1e35346f278>
```



Observations from the plots

1. Gender has 3 negative that is missing values.

2. The Histogram for ages is plotted to find the distribution of ages among the patients and it was observed that the maximum and the minimum values lie between 90 and 15 respectively, and majority of patients belong to the 60-90 age group.

3. ICUType vs Age jointplot shows that there are more patients in Medical ICU (ICUType 3) and patients above age 60 constitute the majority.

4. Box plots of Weight and Height are plotted against their age-group (interval of 10) to find the outliers and it was observed that there are missing values in both heights and weights, in addition to that, uncommon values are also present which has to be handled.

Static Data Analysis & Imputation - Handling Uncommon and Missing Values

In [149]:
```
#-----Handling the missing and incorrect values in the static data

#-----Finding the most frequent values (mode) to impute the missing data.
gender_mode = float(df_static['Gender'].mode())
ICUType_mode = float(df_static['ICUType'].mode())

#-----Finding the average (mean) of these columns to impute the missing data.
age_avg = float(df_static['Age'].mean())
weight_avg = float(df_static['Weight'].mean())
height_avg = float(df_static['Height'].mean())

for c in df_static:
    x = df_static[c]
    #-----'Age' greater than 100 is considered abnormal, such values are imputed with
91.4 (max_value)
    if c == 'Age':
        idx = x > 100
        df_static.loc[idx,c] = 91.4

    #----'Gender' a binary column can have only values 0 and 1. If anyother number is
present,then we impute with mode.
    if c == 'Gender':
        idx = x < 0
        df_static.loc[idx, c] = gender_mode

        idx = x > 1
        df_static.loc[idx, c] = gender_mode
    #----'Weight' has a nominal range between 35 and 300.Anything outer this region i
s considered abnormal and imputed
   #with mean value.
    elif c == 'Weight':
        idx = x < 35
        df_static.loc[idx, c] = weight_avg

        idx = x > 300
        df_static.loc[idx, c] = weight_avg

    #-----'Height' imputation.
    elif c == 'Height':
        idx = x < 0
        df_static.loc[idx,c] = height_avg

        idx = x < 10          #---- '10' is too little a value for height and it must
be wrongly entered.To make it reliable we multiply with 100.
        df_static.loc[idx, c] = df_static.loc[idx, c] * 100

        idx = x < 25          #----- 18 -> 180 (The same condition as above applies f
or this)
        df_static.loc[idx, c] = df_static.loc[idx, c] * 10

        idx = x < 100         #----- 81.8 -> 180 (inch -> cm) For converting units fr
om inches to cm
        df_static.loc[idx, c] = df_static.loc[idx, c] * 2.2

        idx = x > 1000        #----- 1800 -> 180 Any value greater than 1000 is too l
arge to be height, so minimizing by multiplying with 0.1.
        df_static.loc[idx, c] = df_static.loc[idx, c] * 0.1

        idx = x > 250         #----- 400 -> 157 Same condition as above.
        df_static.loc[idx, c] = df_static.loc[idx, c] * 0.3937

    #----'ICU Type' a categorical column. If anyother number other than 1,2,3,4 is pr
esent,then we impute with mode.
    elif c == 'ICUType':
        idx = x < 1
        df_static.loc[idx, c] = ICUType_mode
        idx = x > 4
        df_static.loc[idx,c]= ICUType_mode
```

In [150]:
```
#----- One-hot Encoding for Categorical Feature (ICUType)

df_static = df_static.join(pd.get_dummies(df_static['ICUType'], prefix='ICUType'))
df_static.drop(['ICUType'],axis =1,inplace=True)
df_static.drop(['age_group'],axis =1,inplace=True)
df_static.head()
```

Out[150]:

	Age	Gender	Height	RecordID	Weight	ICUType_1.0	ICUType_2.0	ICUType_3.0	ICUTy
Record_ID									
132539	54.0	0.0	195.622845	132539.0	74.75629	0	0	0	
132540	76.0	1.0	175.300000	132540.0	76.00000	0	1	0	
132541	44.0	0.0	195.622845	132541.0	56.70000	0	0	1	
132543	68.0	1.0	180.300000	132543.0	84.60000	0	0	1	
132545	88.0	0.0	195.622845	132545.0	74.75629	0	0	1	

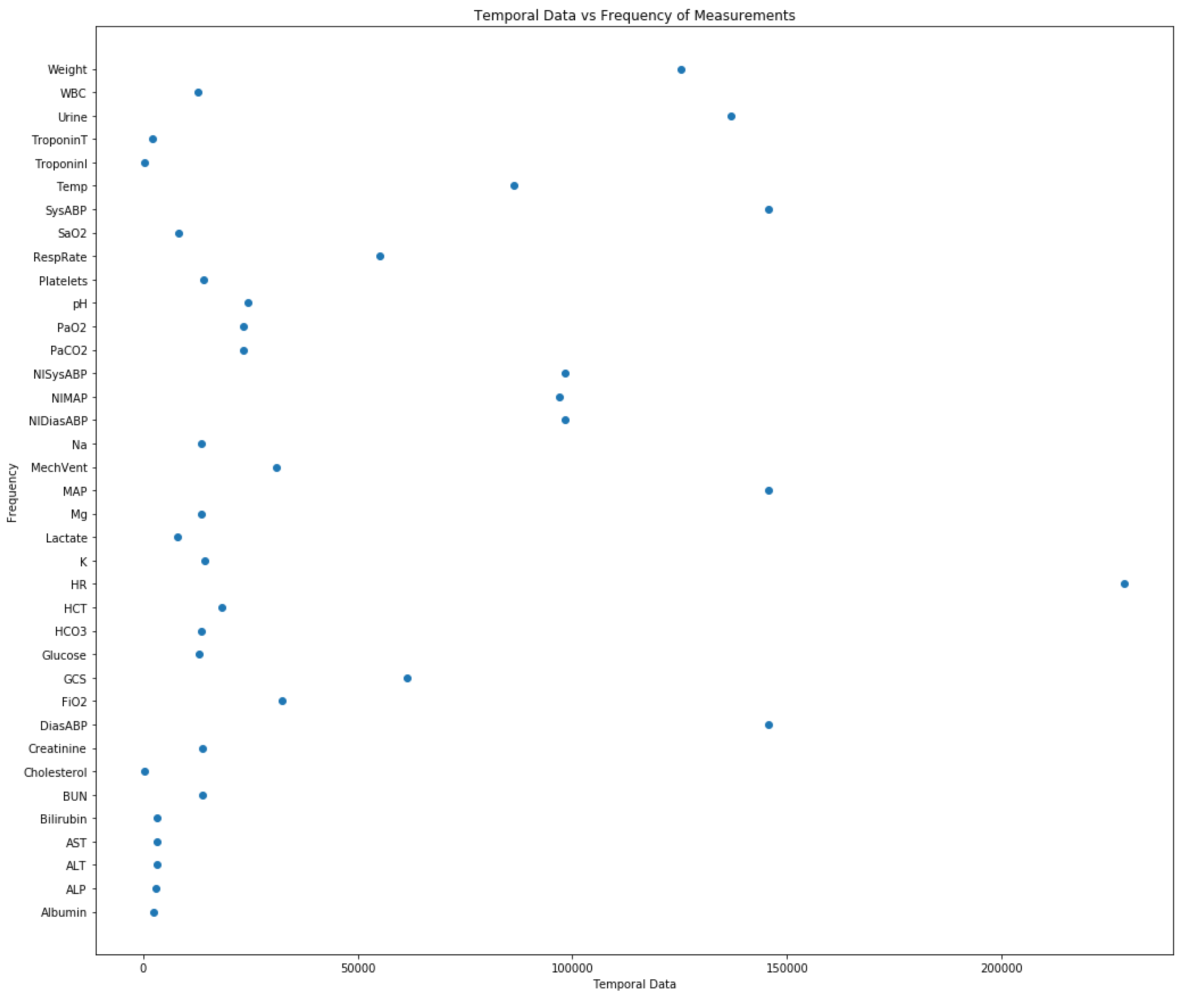
Temporal Data Analysis

In [151]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1733980 entries, 6 to 1757979
Data columns (total 4 columns):
Time         object
Parameter    object
Value        object
Record_ID    object
dtypes: object(4)
memory usage: 66.1+ MB
```

Temporal Data Analysis - Plotting Temporal Data to find Measurement Frequency

In [152]:
```
#----- Feature Engineering for Temporal Data
df['Value'] = df['Value'].astype('float')


#----- Grouping based on the record_id
grp = df.groupby('Record_ID')


dummy = pd.DataFrame(columns=temporal_feat)
dummy.loc[0] = [0 for i in range(37)]

for i in grp.groups.keys():
    x=grp.get_group(i)['Parameter'].value_counts()
    dummy[x.index]  += x


plt.figure(9,figsize = (17,15))
plt.scatter(dummy.loc[0],temporal_feat)
plt.xlabel("Temporal Data")
plt.ylabel("Frequency")
plt.title("Temporal Data vs Frequency of Measurements")
plt.show()
```



Observations from the plots

1. Based on the above group, the following temporal data are measured frequently than other temporal data:

Urine	Temp	SysABP	SaO2	RespRate	NISysABP	NIMAP	NIDiasABP	MAP	HR	GCS	DiasABP

2. So, **First,Last,Min,Max** Measurements of the above preceded temporal features are extracted to predict mortality rate and lenght of the stay.
3. And for the remainining temporal features,**First and Last values** are extracted

In [153]:
```
#----- Temporal features, recorded with higher frequency
temporal_feat_1 = ['DiasABP', 'GCS', 'HR', 'MAP','NIDiasABP', 'NIMAP',
                   'NISysABP', 'RespRate', 'SaO2', 'Temp','Urine' ]
df_tmp = df.loc[df['Parameter'].isin(temporal_feat_1)]
df_tmp = df_tmp.groupby(['Record_ID', 'Parameter'])['Value']

X = df_static.copy()

#----- Getting First, Last, Min and Max for temporal features recorded with higher frequency
for i in ['First','Last','Min','Max']:
    if(i=='First'):
        X_tmp = df_tmp.first()
    elif(i=='Last'):
        X_tmp = df_tmp.last()
    elif(i=='Min'):
        X_tmp = df_tmp.min()
    elif(i=='Max'):
        X_tmp = df_tmp.max()

#----- Changing patient records to rows
    X_tmp = X_tmp.reset_index()
    X_tmp = X_tmp.pivot(index='Record_ID',columns='Parameter',values='Value')
    X_tmp.columns = [j + '_' + i for j in X_tmp.columns]

#----- Adding Temporal features with static
    X = X.merge(X_tmp, how='left' , left_index=True, right_index=True )

X.head()
```

Out[153]:

	Age	Gender	Height	RecordID	Weight	ICUType_1.0	ICUType_2.0	ICUType_3.0	ICUTy
Record_ID									
132539	54.0	0.0	195.622845	132539.0	74.75629	0	0	0	
132540	76.0	1.0	175.300000	132540.0	76.00000	0	1	0	
132541	44.0	0.0	195.622845	132541.0	56.70000	0	0	1	
132543	68.0	1.0	180.300000	132543.0	84.60000	0	0	1	
132545	88.0	0.0	195.622845	132545.0	74.75629	0	0	1	

5 rows × 53 columns

Feature Extraction from Temporal Data - Temporal Features with Low Frequency Measurements

In [154]:
```
#----- Temporal features, recorded with lower frequency
temporal_feat_2 = ['Albumin', 'ALP', 'ALT', 'AST', 'Bilirubin', 'BUN', 'Cholesterol', 'Creatinine', 'FiO2', 'HCO3',
                   'HCT', 'K', 'Lactate', 'Mg', 'Na','PaCO2', 'PaO2', 'pH', 'Platelets', 'SysABP', 'TroponinI',
                   'TroponinT','WBC']

df_tmp = df.loc[df['Parameter'].isin(temporal_feat_2)]
df_tmp = df_tmp.groupby(['Record_ID', 'Parameter'])['Value']

#----- Getting First, Last, Min and Max for temporal features recorded with lower frequency
for i in ['First','Last']:
    if(i=='First'):
        X_tmp = df_tmp.first()
    elif(i=='Last'):
        X_tmp = df_tmp.last()

#----- Changing patient records to rows
    X_tmp = X_tmp.reset_index()
    X_tmp = X_tmp.pivot(index='Record_ID',columns='Parameter',values='Value')
    X_tmp.columns = [j + '_' + i for j in X_tmp.columns]

#----- Adding features with previously extracted ones
    X = X.merge(X_tmp, how='left' , left_index=True, right_index=True )

X.head()
```

Out[154]:

	Age	Gender	Height	RecordID	Weight	ICUType_1.0	ICUType_2.0	ICUType_3.0	ICUTy
Record_ID									
132539	54.0	0.0	195.622845	132539.0	74.75629	0	0	0	
132540	76.0	1.0	175.300000	132540.0	76.00000	0	1	0	
132541	44.0	0.0	195.622845	132541.0	56.70000	0	0	1	
132543	68.0	1.0	180.300000	132543.0	84.60000	0	0	1	
132545	88.0	0.0	195.622845	132545.0	74.75629	0	0	1	

5 rows × 99 columns

In [155]:
```
print(X.shape)
```

```
(4000, 99)
```

Feature Extraction from Temporal Data - Ploting of Mechanical Ventialtion (Temporal_Binary data)

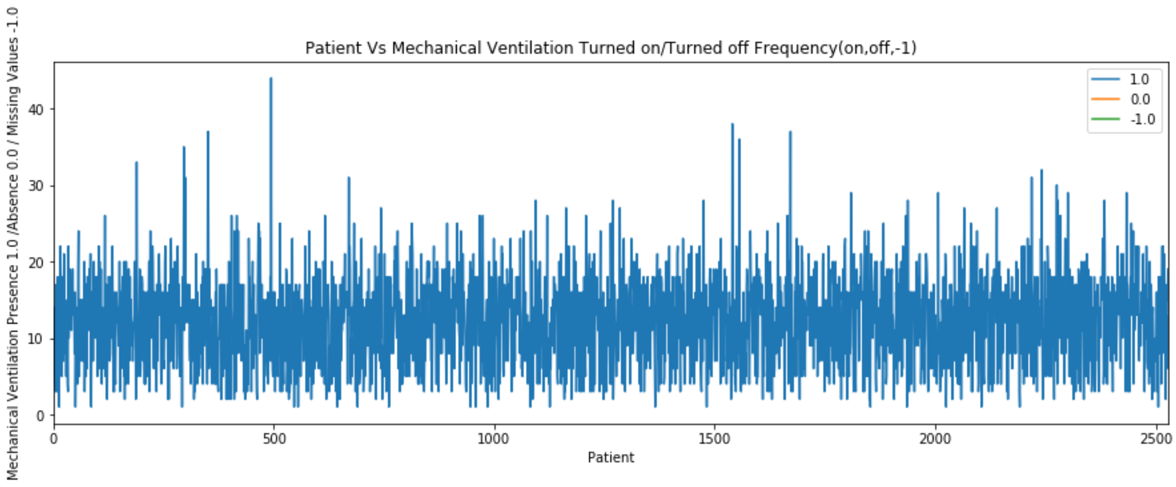
Since **Mechanical Ventilation is a continous time value**, but it is indicated as YES or NO. So, the following code block shows the conversion of YES or NO (Binary) to Continuous Value using "Time" Feature

To find ventilation is

1. *Continuously on*
2. *Continuously off*
3. *Switching between on and off*

In [156]:
```
#----- Mechanical Ventilation YES or NO to Coninuous Value
df_tmp = df.loc[df['Parameter']=="MechVent"]
df_tmp = df_tmp.groupby('Record_ID')

#----- Finding is if the ventilation is " on for sometimes and off for sometimes" ,
#-----                                  " continuously on" ,
#-----                                  " continuously off"

interval = pd.DataFrame(columns = [1.0,0.0,-1])
count = 0
for i in df_tmp.groups.keys():          #------ Grouped based on RecordID
    interval.loc[count] = df_tmp.get_group(i)['Value'].value_counts()
    count = count+1

#interval = interval.T
interval.plot(figsize = (15,5), title = "Patient Vs Mechanical Ventilation Turned on/Turned off Frequency(on,off,-1)")
plt.xlabel("Patient")
plt.ylabel("Mechanical Ventilation Presence 1.0 /Absence 0.0 / Missing Values -1.0")
```

Out[156]:
```
Text(0, 0.5, 'Mechanical Ventilation Presence 1.0 /Absence 0.0 / Missing Values -1.0')
```



Observations from the plots

Mechanical ventilation is always on for ~2600 Patients and for the rest of the patient mechanical ventilation is not used

In [157]:
```
#----- Finding min and converting string 00:00 to hours format
start_time = df_tmp[['Time']].min()
start_time['Time'] = (start_time['Time'].str.split(':').str[0].astype(int))*60 + start_time['Time'].str.split(':').str[1].astype(int)
start_time.columns = ['VentilationStartTime']

#----- Adding Ventilation starting time is one of the features
X = X.merge(start_time, how='left', left_index=True, right_index=True)

#----- Finding max , converting string 00:00 to hours format , duration
time = df_tmp[['Time']].max()
time['Time'] = (time['Time'].str.split(':').str[0].astype(int))*60 + time['Time'].str.split(':').str[1].astype(int)
time['Time'] = time['Time'] - start_time['VentilationStartTime']
time.columns = ['duration']

#----- Adding Ventilation starting time is one of the features
X = X.merge(time, how='left', left_index=True, right_index=True)
print(X.shape)
X.head()
```

```
(4000, 101)
```

Out[157]:

	Age	Gender	Height	RecordID	Weight	ICUType_1.0	ICUType_2.0	ICUType_3.0	ICUTy
Record_ID									
132539	54.0	0.0	195.622845	132539.0	74.75629	0	0	0	
132540	76.0	1.0	175.300000	132540.0	76.00000	0	1	0	
132541	44.0	0.0	195.622845	132541.0	56.70000	0	0	1	
132543	68.0	1.0	180.300000	132543.0	84.60000	0	0	1	
132545	88.0	0.0	195.622845	132545.0	74.75629	0	0	1	

5 rows × 101 columns

1.3 Temporal Data Imputation - Handling Missing Values/Non-recorded parameters to 0

In [158]:
```
#----- Finding Number of records having NAN in each features
print(X.isna().sum())

#----- Finding Missing records in each features
for i in X:
    if( (X[i]<0).sum() > 0):
        print(i,(X[i]<0).sum())

#----- Replacing Missing values and NAN to 0
X = X.fillna(0)
```

```
Age                       0
Gender                    0
Height                    0
RecordID                  0
Weight                    0
ICUType_1.0               0
ICUType_2.0               0
ICUType_3.0               0
ICUType_4.0               0
DiasABP_First          1201
GCS_First                64
HR_First                 63
MAP_First              1208
NIDiasABP_First         517
NIMAP_First             519
NISysABP_First          507
RespRate_First         2899
SaO2_First             2208
Temp_First               64
Urine_First             117
DiasABP_Last           1201
GCS_Last                 64
HR_Last                  63
MAP_Last               1208
NIDiasABP_Last          517
NIMAP_Last              519
NISysABP_Last           507
RespRate_Last          2899
SaO2_Last              2208
Temp_Last                64
                       ...
SysABP_First           1201
TroponinI_First        3795
TroponinT_First        3137
WBC_First                73
pH_First                960
ALP_Last               2310
ALT_Last               2279
AST_Last               2275
Albumin_Last           2385
BUN_Last                 64
Bilirubin_Last         2282
Cholesterol_Last       3695
Creatinine_Last          64
FiO2_Last              1283
HCO3_Last                76
HCT_Last                 64
K_Last                   96
Lactate_Last           1817
Mg_Last                 103
Na_Last                  75
PaCO2_Last              977
PaO2_Last               977
Platelets_Last           68
SysABP_Last            1201
TroponinI_Last         3795
TroponinT_Last         3137
WBC_Last                 73
pH_Last                 960
VentilationStartTime   1471
duration               1471
Length: 101, dtype: int64
Temp_First 4
Temp_Last 2
Temp_Min 24
```

Writing Design Matrix to csv file

Train.csv holds the data for design matrix 1

In [159]:
```
X.to_csv("Train.csv")
```

XGB Classification for "In-Hospital Death" Prediction

XGB Classification Model

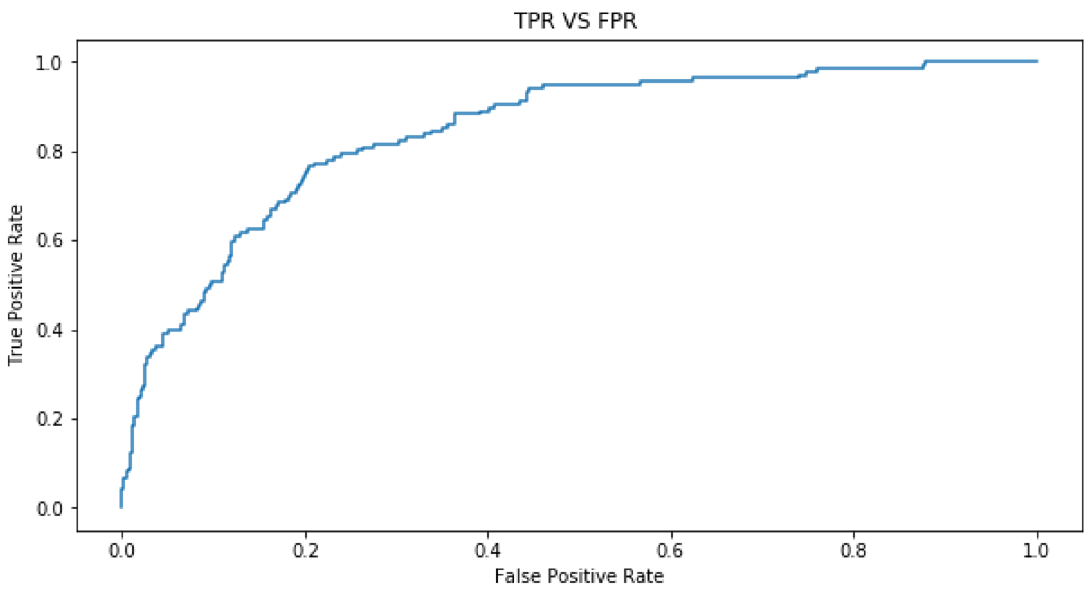
Model Parameters
            n_estimator = 550,
            min_child_weight = 5 obtained after randomized grid search over the entire dat
aset

In [160]:
```
Mean_auc = []
label = [[1,2,3,4],[1,2,4,3],[1,3,4,2],[2,3,4,1]]

#----- XGBClassifier
#----- Iterating through all the folds in the below sequence

#Iteration 1:  Training data: fold 1, 2, 3, Test data: fold 4
#Iteration 2:  Training data: fold 1, 2, 4, Test data: fold 3
#Iteration 3:  Training data: fold 1, 3, 4, Test data: fold 2
#Iteration 4:  Training data: fold 4, 2, 3, Test data: fold 1

for i in range(0,4):
    Y_train = pd.concat([pd.read_csv("Fold" + str(label[i][0]) + "_Outcomes.csv"),
                     pd.read_csv("Fold" + str(label[i][1])  + "_Outcomes.csv"),
                     pd.read_csv("Fold" + str(label[i][2]) + "_Outcomes.csv")])

    Y_test = pd.read_csv("Fold" + str(label[i][3]) + "_Outcomes.csv")

    X_train = pd.read_csv("Train.csv")

    #----- Load the matching records from Y_train (Record_ID) in X_train and unmatche
d records in X_test

    X_test = X_train.loc[~X_train['RecordID'].isin(Y_train['RecordID'])]
    X_train = X_train.loc[X_train['RecordID'].isin(Y_train['RecordID'])]
    #-------Dropping the insignificant features
    X_test.drop(['Record_ID','RecordID'],axis=1,inplace=True)
    X_train.drop(['Record_ID','RecordID'],axis=1,inplace=True)

    model = XGBClassifier(n_estimator = 550,min_child_weight=5)

    #------ In-hospital_death => Classifier target variable

    model.fit(X_train, Y_train['In-hospital_death'])
    Mean_auc.append(roc_auc_score(Y_test['In-hospital_death'],model.predict_proba(X_t
est)[:,1]))
    print("ROC_AUC Score Fold",label[i][3],"as Testing Data: ",Mean_auc[i])

print("Mean ROC_AUC Score",np.mean(Mean_auc))
```

```
ROC_AUC Score Fold 4 as Testing Data:  0.8517058486238532
ROC_AUC Score Fold 3 as Testing Data:  0.8483206933911159
ROC_AUC Score Fold 2 as Testing Data:  0.8522237025758151
ROC_AUC Score Fold 1 as Testing Data:  0.8436053240740742
Mean ROC_AUC Score 0.8489638921662146
```

In [161]:
```
Y_pred = model.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(Y_test['In-hospital_death'] , Y_pred)

auc = roc_auc_score(Y_test['In-hospital_death'] , Y_pred)
plt.figure(9,figsize = (10,5))
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("TPR VS FPR ")
plt.plot(fpr,tpr,label="data, auc="+str(auc))
```

Out[161]:
```
[<matplotlib.lines.Line2D at 0x1e37e320d30>]
```



1.From the above graph, Threshold value at the top left corner should be choosen such that False Positive is less at the same time True Positive is high.
2. Since, we are dealing with Mortality prediction, False Positive should be as low as possible

XGB Regression for "Length of the Stay" Prediction

XGB Regressor With Scalar

```
Model Parameters
            n_estimators=150,
            max_depth=6,
            subsample=0.6,
            learning_rate=0.03,
            gamma=0,
            colsample_bytree=0.3,
            silent=False,
            objective="reg:linear" obtained after randomized grid search over the entire d
ataset
```

In [163]:
```
Mean_rms = []
label = [[1,2,3,4],[1,2,4,3],[1,3,4,2],[2,3,4,1]]
preprocesser = Pipeline(steps=[('scaler',StandardScaler())])

#----- XGBRegressor with scaler
#----- Iterating through all the folds in the below sequence

#Iteration 1:  Training data: fold 1, 2, 3, Test data: fold 4
#Iteration 2:  Training data: fold 1, 2, 4, Test data: fold 3
#Iteration 3:  Training data: fold 1, 3, 4, Test data: fold 2
#Iteration 4:  Training data: fold 4, 2, 3, Test data: fold 1

for i in range(0,4):
    Y_train = pd.concat([pd.read_csv("Fold" + str(label[i][0]) + "_Outcomes.csv"),
                     pd.read_csv("Fold" + str(label[i][1])  + "_Outcomes.csv"),
                     pd.read_csv("Fold" + str(label[i][2]) + "_Outcomes.csv")])

    Y_test = pd.read_csv("Fold" + str(label[i][3]) + "_Outcomes.csv")

    X_train = pd.read_csv("Train.csv")

    #----- Load the matching records from Y_train (Record_ID) in X_train and unmatche
d records in X_test
    X_test = X_train.loc[~X_train['RecordID'].isin(Y_train['RecordID'])]
    X_train = X_train.loc[X_train['RecordID'].isin(Y_train['RecordID'])]

    #-------Dropping the insignificant features
    X_test.drop(['Record_ID','RecordID'],axis=1,inplace=True)
    X_train.drop(['Record_ID','RecordID'],axis=1,inplace=True)

    X_train_scaled = preprocesser.fit_transform(X_train,Y_train["Length_of_stay"])
    X_test_scaled = preprocesser.transform(X_test)


    reg_xgb = XGBRegressor(n_estimators=150, max_depth=6, subsample=0.6,learning_rate
=0.03, gamma=0,
                           colsample_bytree=0.3,silent=False, objective="reg:linear")
    reg_xgb.fit(X_train_scaled, Y_train["Length_of_stay"])
    Y_pred = reg_xgb.predict(X_test_scaled)
    Mean_rms.append(np.sqrt(mean_squared_error(Y_test["Length_of_stay"], Y_pred)))
    print("RMS Fold",label[i][3],"as Testing Data: ",Mean_rms[i])

print("\nMean RMS Score",np.mean(Mean_rms))
```

```
[07:38:52] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated
in favor of reg:squarederror.
RMS Fold 4 as Testing Data:  10.190987820572294
[07:38:55] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated
in favor of reg:squarederror.
RMS Fold 3 as Testing Data:  11.007219691107283
[07:38:58] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated
in favor of reg:squarederror.
RMS Fold 2 as Testing Data:  10.940765096467558
[07:39:01] WARNING: src/objective/regression_obj.cu:152: reg:linear is now deprecated
in favor of reg:squarederror.
RMS Fold 1 as Testing Data:  13.071284572229077

Mean RMS Score 11.302564295094053
```

End of Notebook